A Highly Parallelizable Newton-type Method for Nonlinear Model Predictive Control

Deng, Haoyang; Ohtsuka, Toshiyuki

A Highly Parallelizable Newton-type Method for Nonlinear Model Predictive Control ⋆

Haoyang Deng * **Toshiyuki Ohtsuka** *

* *Department of Systems Science, Graduate School of Informatics, Kyoto University, Japan*
(e-mail: {deng, ohtsuka}@sys.i.kyoto-u.ac.jp)

Abstract: We propose a highly parallelizable Newton-type method for nonlinear model predictive control by exploiting the particular structure of the associated Karush-Kuhn-Tucker conditions. These equations are approximately decoupled into single step subproblems along the prediction horizon for parallelization. The coupling variable of each subproblem is approximated toward its optimal value by a simple but effective method in every iteration. The proposed algorithm is applied to control a quadrotor. The numerical simulation results show that the proposed algorithm is highly parallelizable and converges with only a few iterations even to a high accuracy. The proposed method is also shown to be faster compared with several state-of-the-art algorithms.

Keywords: Nonlinear model predictive control, parallel algorithm, real-time algorithm

1. INTRODUCTION

Nonlinear model predictive control (NMPC), due to its ability to handle nonlinear dynamics and constraints explicitly, has gained a lot of attention. However, it requires an optimal control problem (OCP) to be solved within a specified time interval at each sampling instant, which is computationally expensive for problems with large dimensionalities and long prediction horizons, and thus precludes its broader applications.

To solve the underlying OCP in real time, fast algorithms have been developed. Generally, real-time algorithms tailored to NMPC can be categorized into either the first-order methods or the Newton-type methods depending on how each iteration is performed. First-order methods (see e.g., Kalmari et al., 2015; Kouzoupis et al., 2015; Graichen and Käpernick, 2012) can generally be parallelized but are restricted to a certain class of problems, such as problems with only input bounds. Many of the first-order methods depend on offline preparations such as decomposition of the Hessian matrix to speed up the online calculations. However, in NMPC, the Hessian matrix is generally not constant, which makes the online decomposition computationally expensive. Newton-type methods, such as the real-time iteration (RTI) scheme (Diehl et al., 2002) and the continuation/generalized minimal residual (C/GMRES) method (Ohtsuka, 2004), converge faster than the first-order methods and are able to deal with more general problems. Still, performing one iteration is computationally more expensive, and the degree of parallelism is lower than in the first-order methods.

⋆ This work was partly supported by JSPS KAKENHI Grant Number 15H02257.

The demand for parallel algorithms tailored for NMPC rapidly increases with the development of parallel hardware, such as multi-core processors, graphics processing units (GPUs), and field-programmable gate arrays (FPGAs), so several parallel algorithms for NMPC have been proposed. Newton's method based on parallel Riccati recursion (see Frasch et al., 2015; Nielsen and Axehill, 2015) can achieve computational complexity of $\mathcal{O}(\log N)$, where N is the number of discretization grids on the horizon. The advanced multi-step NMPC proposed by Yang and Biegler (2013), which is parallelized along the time axis, solves the NMPC problems in background multiple sampling times in advance concurrently based on the predicted states and corrects the predicted input based on sensitivity. Particle swarm optimization, due to its inherent parallelism, has been implemented on an FPGA for NMPC by Xu et al. (2016). An augmented Lagrangian method tailored to OCP is proposed by Kouzoupis et al. (2016), where subproblems along the prediction steps are solved concurrently, and a centralized consensus quadratic programming (QP) problem is solved to update the dual variables in each iteration. However, the trade-off between the size of the consensus QP and the degree of parallelism restricts its speed-up according to Amdahl's law (see Amdahl, 1967).

In this paper, we propose a novel highly parallelizable Newton-type method for NMPC. First, the continuous OCP is discretized with the implicit Euler method so that the Karush-Kuhn-Tucker (KKT) conditions for the discretized problem can be split into linearly coupled subproblems. Second, a sequential method named the backward correction method is formulated. The backward correction method is proved to be exactly identical to Newton's method but with a clearer structure for parallelization. In the backward correction method, the coupling variable in each subproblem is calculated recursively in a

backward manner, which takes a lot of time. We therefore propose a reasonable approximation for the coupling variable to break down the recursion, so that the calculation can be done in parallel. Finally, the proposed method is applied to control a quadrotor. The numerical simulation shows that the proposed method is highly parallelizable and converges to the specified tolerance with only a few iterations.

This paper is organized as follows. First, the NMPC problem is formulated in Section 2. The backward correction method is formulated and proved to be exactly identical to Newton's method in Section 3. The proposed method and its fully parallelized algorithm are given in Section 4. Section 5 describes the numerical experiment. Finally, conclusions and future work are summarized in Section 6.

2. PROBLEM FORMULATION

In this section, the discretized NMPC problem is formulated, and its corresponding first-order conditions for optimality are given.

2.1 Discretized NMPC

Throughout this paper, the discretized NMPC problem to be considered is given by:

$$\begin{aligned} \min_{X,U} \ & \sum_{i=1}^{N} L(u_i, x_i, p_i)\Delta\tau \\ s.t. \quad & x_0 = \bar{x}_0, \\ & x_i = x_{i-1} + f(u_i, x_i, p_i)\Delta\tau, \\ & C(u_i, x_i, p_i)\Delta\tau = 0, \quad \text{for all } i \in \{1, \cdots, N\}. \end{aligned} \tag{1}$$

where $\Delta\tau$ denotes the discretization step size, N the number of discretization grids, $x_i \in \mathbb{R}^{n_x}$ the state, $u_i \in \mathbb{R}^{n_u}$ the control input, $p_i \in \mathbb{R}^{n_p}$ the given parameter, and $\bar{x}_0$ the initial state. We introduced vectors of variables on the horizon $X = (x_0, x_1, x_2, \cdots, x_N)$ and $U = (u_1, u_2, \cdots, u_N)$. We adopted the implicit Euler method to discretize the dynamics. Note that the reference tracking, terminal cost function, and time varying dynamics can all be achieved by the parameter p. Problems with inequality constraints can be converted into equality constrained problems by introducing dummy inputs (Ohtsuka, 2004) or using barrier functions.

2.2 KKT conditions

Define the Hamiltonian $H(\lambda, \mu, u, x, p)$ by

$$H(\lambda, \mu, u, x, p) := L(u, x, p) + \lambda^T f(u, x, p) + \mu^T C(u, x, p),$$

where $\lambda \in \mathbb{R}^{n_x}$ and $\mu \in \mathbb{R}^{n_\mu}$ are the Lagrange multipliers for the difference equation and equality constraint, respectively. The sequences of the optimal control input $\{u_i^*\}_{i=1}^N$, state $\{x_i^*\}_{i=1}^N$, costate $\{\lambda_i^*\}_{i=1}^N$ and multiplier $\{\mu_i^*\}_{i=1}^N$ satisfy the following nonlinear algebraic equations:

$$\begin{aligned} & x_{i-1}^* - x_i^* + f(u_i^*, x_i^*, p_i)\Delta\tau = 0, \ i = 1, \cdots, N, \\ & C(u_i^*, x_i^*, p_i)\Delta\tau = 0, \ i = 1, \cdots, N, \\ & H_u^T(\lambda_i^*, \mu_i^*, u_i^*, x_i^*, p_i)\Delta\tau = 0, \ i = 1, \cdots, N, \\ & \lambda_{i+1}^* - \lambda_i^* + H_x^T(\lambda_i^*, \mu_i^*, u_i^*, x_i^*, p_i)\Delta\tau = 0, \ i = 1, \cdots, N, \end{aligned} \tag{2}$$

given $x_0^* = \bar{x}_0$ and $\lambda_{N+1}^* = 0$. Here, $H_u := \partial H/\partial u$ and $H_x := \partial H/\partial x$. Equations (2) are derived from the KKT conditions and are also known as the discrete-time Euler Lagrange equations, which are the first-order conditions for optimality of problem (1). Note that discretization by the implicit Euler method leads to simple couplings between stages.

2.3 Notations

Define a vector of unknown variables with subscript i as $\mathcal{V}_i := [\lambda_i^T, \mu_i^T, u_i^T, x_i^T]^T \in \mathbb{R}^n$, and a mapping $\mathcal{U}_i(x_{i-1}, \mathcal{V}_i, \lambda_{i+1}) : \mathbb{R}^{n_x} \times \mathbb{R}^n \times \mathbb{R}^{n_x} \to \mathbb{R}^n$ as the left-hand sides of equations (2) with subscript i (stage i), where $n = 2n_x + n_\mu + n_u$. Denote $\mathcal{U} := [\mathcal{U}_1^T, \mathcal{U}_2^T, \cdots, \mathcal{U}_N^T]^T$, $\mathcal{V} := [\mathcal{V}_1^T, \mathcal{V}_2^T, \cdots, \mathcal{V}_N^T]^T$, $\mathcal{U}_{i:j} := [\mathcal{U}_i^T, \mathcal{U}_{i+1}^T, \cdots, \mathcal{U}_j^T]^T$ and $\mathcal{V}_{i:j} := [\mathcal{V}_i^T, \mathcal{V}_{i+1}^T, \cdots, \mathcal{V}_j^T]^T$. Then, $\mathcal{U}(\mathcal{V}) = 0$ is the compact form of equations (2) and denote by $\mathcal{V}^*$ its solution. For a variable v, v^k stands for the value of v at the k-th iteration. Denote $J(\mathcal{V}) := \mathcal{U}'(\mathcal{V})$ as the Jacobian matrix of $\mathcal{U}$ with respect to $\mathcal{V}$. Define $\mathcal{U}_i^k := \mathcal{U}_i(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^k)$ and $J_{i:j}^k := \frac{\partial \mathcal{U}_{i:j}}{\partial \mathcal{V}_{i:j}}|_{(x_{i-1}^k, \mathcal{V}_{i:j}^k, \lambda_{j+1}^k)}$ for the sake of simplicity. For sets of indexes α and β, $A[\alpha, \beta]$ is the submatrix of $A \in \mathbb{R}^{m \times n}$ with rows indexed by α and columns indexed by β. Define $[A]_k^U := A[1:k, 1:k]$. The symbol $\|\cdot\|$ denotes the Euclidean norm for a vector and the p-norm for a matrix, respectively.

3. BACKWARD CORRECTION METHOD

3.1 Motivation

Newton's method is practical and powerful for solving the nonlinear algebraic equation $\mathcal{U}(\mathcal{V}) = 0$, as it is simple to implement and converges quadratically under mild conditions. The full-step Newton's method performs the following iteration (3) starting from an initial guess $\mathcal{V}^0$ that is sufficiently close to $\mathcal{V}^*$:

$$\mathcal{V}^{k+1} = \mathcal{V}^k - J(\mathcal{V}^k)^{-1}\mathcal{U}(\mathcal{V}^k), \ k = 0, 1, \cdots. \tag{3}$$

However, solving the underlying linear equation in Newton's iteration is computationally expensive. The computational complexity can be either $\mathcal{O}(Nn^3)$ by exploiting the banded structure of $J(\mathcal{V})$, or $\mathcal{O}(N^3(n_u+n_\mu)^3)$ by condensing. Although methods by parallel Riccati recursion (see Frasch et al., 2015; Nielsen and Axehill, 2015) can have complexities of $\mathcal{O}(\log(N)n^3)$, the degree of parallelism is limited.

Notice that the KKT conditions at stage i are coupled with the neighboring stages linearly, that is, x_{i-1} and λ_{i+1} enter the equation $\mathcal{U}_i(x_{i-1}, \mathcal{V}_i, \lambda_{i+1}) = 0$ linearly. Equation $\mathcal{U}_i(x_{i-1}, \mathcal{V}_i, \lambda_{i+1}) = 0$ can then be formulated as the KKT condition for a single step OCP with an initial state x_{i-1} and extra penalty $\lambda_{i+1}^T x_i$. As a result, solving $\mathcal{U}(\mathcal{V}) = 0$ can be parallelized by solving a series of equations $\mathcal{U}_i(x_{i-1}^*, \mathcal{V}_i, \lambda_{i+1}^*) = 0$ with respect to $\mathcal{V}_i$, $i = 1, \cdots, N$, if x_{i-1}^* and λ_{i+1}^* are given for each stage in advance. This is similar to the idea of dynamic programming that any part of the optimal trajectory itself must be optimal. Unfortunately, x_{i-1}^* and λ_{i+1}^* cannot be known in advance, and only suboptimal values are given in general.

Consider the following iteration (4) for approximately solving $\mathcal{U}_i(\tilde{x}_{i-1}, \mathcal{V}_i, \tilde{\lambda}_{i+1}) = 0$:

$$\mathcal{V}_i^{k+1} = \mathcal{V}_i^k - \left(\left. \frac{\partial \mathcal{U}_i}{\partial \mathcal{V}_i} \right|_{(\tilde{x}_{i-1}, \mathcal{V}_i^k, \tilde{\lambda}_{i+1})} \right)^{-1} \mathcal{U}_i(\tilde{x}_{i-1}, \mathcal{V}_i^k, \tilde{\lambda}_{i+1}). \tag{4}$$

where $\tilde{x}_{i-1}$ and $\tilde{\lambda}_{i+1}$ are the estimates of x_{i-1}^* and λ_{i+1}^*, respectively. One of the simplest methods takes $\tilde{x}_{i-1} = x_{i-1}^k$ and $\tilde{\lambda}_{i+1} = \lambda_{i+1}^k$. This method has complexity $\mathcal{O}(n^3)$ in parallel but it may diverge. The Gauss-Seidel scheme performs (4) recursively from $i = 1$ to N, with $\tilde{x}_{i-1} = x_{i-1}^{k+1}$ and $\tilde{\lambda}_{i+1}$, for example, estimated by a coarse-grained high-level controller in advance, as proposed by Zavala (2016). The convergence of the Gauss-Seidel scheme cannot be guaranteed either, unless λ_{i+1}^* is well estimated, which is computationally expensive.

Next, we show a new method that estimates λ_{i+1}^* on the basis of the current k-th iteration's information. In the Gauss-Seidel method, stages $2, \cdots, N$ are iterated after the iteration of stage 1. Namely, $\mathcal{V}_{2:N}^{k+1}$ can be regarded as a function of x_1 as follows:

$$\mathcal{V}_{2:N}^{k+1}(x_1) = \mathcal{V}_{2:N}^k - \left(\left. \frac{\partial \mathcal{U}_{2:N}}{\partial \mathcal{V}_{2:N}} \right|_{(x_1, \mathcal{V}_{2:N}^k, \lambda_{N+1})} \right)^{-1} \mathcal{U}_{2:N}(x_1, \mathcal{V}_{2:N}^k, \lambda_{N+1}). \tag{5}$$

As x_1 enters $\mathcal{U}_{2:N}$ linearly, the following equation holds:

$$\mathcal{U}_{2:N}(x_1, \mathcal{V}_{2:N}^k, \lambda_{N+1}) = \mathcal{U}_{2:N}(x_1^k, \mathcal{V}_{2:N}^k, \lambda_{N+1}) + \begin{bmatrix} (x_1 - x_1^k)^T & 0^T \end{bmatrix}^T.$$

Therefore, the Jacobian of $\mathcal{U}_{2:N}$ with respect to $\mathcal{V}_{2:N}$ does not depend on x_1, and the update of λ_2 can be extracted from (5) and expressed as

$$\lambda_2^{k+1}(x_1) = \lambda_2^k - \Lambda_2^k(x_1 - x_1^k) - d_{\lambda_2}^k, \tag{6}$$

where

$$\Lambda_2^k = \left[\left(J_{2:N}^k \right)^{-1} \right]_{n_x}^U \in \mathbb{R}^{n_x \times n_x}$$

and

$$d_{\lambda_2}^k = \left(\left(J_{2:N}^k \right)^{-1} \right) [1 : n_x, :] \cdot \mathcal{U}_{2:N}^k \in \mathbb{R}^{n_x}.$$

Although λ_2^* cannot be known in advance, it can be seen as a correction of λ_2^k by (6) so that the iteration of solving $\mathcal{U}_1(x_0, \mathcal{V}_1, \lambda_2^{k+1}(x_1)) = 0$ with respect to $\mathcal{V}_1$ is given by

$$\begin{aligned} \mathcal{V}_1^{k+1} &= \mathcal{V}_1^k - \left(\left. \frac{\partial \mathcal{U}_1}{\partial \mathcal{V}_1} \right|_{(\bar{x}_0, \mathcal{V}_1^k, \lambda_2^{k+1}(x_1^k))} \right)^{-1} \mathcal{U}_1(\bar{x}_0, \mathcal{V}_1^k, \lambda_2^{k+1}(x_1^k)) \\ &= \mathcal{V}_1^k - \left(J_1^k - \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_2^k \end{bmatrix} \right)^{-1} \left(\mathcal{U}_1^k - \begin{bmatrix} 0 \\ d_{\lambda_2}^k \end{bmatrix} \right). \end{aligned}$$

After obtaining x_1^{k+1}, stages $2, \cdots, N$ can be further split and iterated recursively, which results in the following method.

3.2 Algorithm of the backward correction method

It should be noticed that the stages cannot be further split at the last stage because $\lambda_{N+1}^* = 0$ is already known. Therefore, the coupling variable λ_{i+1} for stage i is corrected from $i = N-1$ to 1 recursively in a backward manner. Thus, the resulting method is called the backward correction method, as shown in Algorithm 1.

The computational complexity of the backward correction method is $\mathcal{O}(Nn^3)$, and it can be categorized into the method exploiting the banded structure of Jacobian $J(\mathcal{V})$. A theorem, which shows that the backward correction method is identical to Newton's method, is stated below.

Algorithm 1 k-th iteration of the backward correction method

Input: $\mathcal{V}^k$, $x_0^k = \bar{x}_0$, $\Lambda_{N+1}^k = 0$, and $d_{\lambda_{i+1}}^k = 0$.

Output: $\mathcal{V}^{k+1}$.

for $i = N$ to 1 **do**

Compute:

$$H_i^k := \left(\left. \frac{\partial \mathcal{U}_i}{\partial \mathcal{V}_i} \right|_{\left(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k)\right)} \right)^{-1} = \left(J_i^k - \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_{i+1}^k \end{bmatrix} \right)^{-1}, \tag{7}$$

where

$$\mathcal{U}_i = \mathcal{U}_i\left(x_{i-1}, \mathcal{V}_i, \lambda_{i+1}^{k+1}(x_i)\right) \tag{8}$$

and

$$\lambda_{i+1}^{k+1}(x_i) = \lambda_{i+1}^k - \Lambda_{i+1}^k(x_i - x_i^k) - d_{\lambda_{i+1}}^k. \tag{9}$$

Update:

$$\Lambda_i^k = \left[H_i^k \right]_{n_x}^U \tag{10}$$

and

$$d_{\lambda_i}^k = H_i^k[1 : n_x, :] \cdot \mathcal{U}_i\left(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k)\right). \tag{11}$$

end for

for $i = 1$ to N **do**

Update:

$$\mathcal{V}_i^{k+1} = \mathcal{V}_i^k - H_i^k \cdot \mathcal{U}_i\left(x_{i-1}^{k+1}, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k)\right). \tag{12}$$

end for

Theorem 1. The backward correction method in Algorithm 1 is identical to Newton's method. That is, starting from the same $\mathcal{V}^k$ at the k-th iteration, the updated value by Newton's method denoted by $\mathcal{V}_{(nt)}^{k+1}$ is equal to the value by the backward correction method denoted by $\mathcal{V}_{(bc)}^{k+1}$.

Proof. Define $\Delta\mathcal{V}_{(nt)}^k := \mathcal{V}_{(nt)}^{k+1} - \mathcal{V}^k$ and $\Delta\mathcal{V}_{(bc)}^k := \mathcal{V}_{(bc)}^{k+1} - \mathcal{V}^k$. First, the calculation of $\Delta\mathcal{V}_{(nt)}^k$ is given. Note that stage i is only coupled with its neighboring stages linearly, and the following equations hold:

$$\frac{\partial \mathcal{U}_1}{\partial \mathcal{V}_2} = \frac{\partial \mathcal{U}_2}{\partial \mathcal{V}_3} = \cdots = \frac{\partial \mathcal{U}_{N-1}}{\partial \mathcal{V}_N} = \begin{bmatrix} 0 & 0 \\ I_{n_x} & 0 \end{bmatrix} =: \mathcal{B}_U \in \mathbb{R}^{n \times n},$$

$$\frac{\partial \mathcal{U}_2}{\partial \mathcal{V}_1} = \frac{\partial \mathcal{U}_3}{\partial \mathcal{V}_2} = \cdots = \frac{\partial \mathcal{U}_N}{\partial \mathcal{V}_{N-1}} = \begin{bmatrix} 0 & I_{n_x} \\ 0 & 0 \end{bmatrix} =: \mathcal{B}_L \in \mathbb{R}^{n \times n}.$$

Thus, the Jacobian $J(\mathcal{V})$ is a block tridiagonal matrix with the following form:

$$J(\mathcal{V}) = \begin{bmatrix} J_1 & \mathcal{B}_U & & \\ \mathcal{B}_L & J_2 & \ddots & \\ & \ddots & \ddots & \mathcal{B}_U \\ & & \mathcal{B}_L & J_N \end{bmatrix}.$$

The linear equation $J(\mathcal{V}^k)\Delta\mathcal{V}_{(nt)}^k + \mathcal{U}(\mathcal{V}^k) = 0$ can be solved by block Gaussian elimination, that is, solving

$$\begin{bmatrix} X_1^k & & & \\ \mathcal{B}_L & X_2^k & & \\ & \ddots & \ddots & \\ & & \mathcal{B}_L & X_N^k \end{bmatrix} \begin{bmatrix} \Delta\mathcal{V}_{1(nt)}^k \\ \Delta\mathcal{V}_{2(nt)}^k \\ \vdots \\ \Delta\mathcal{V}_{N(nt)}^k \end{bmatrix} = - \begin{bmatrix} Y_1^k \\ Y_2^k \\ \vdots \\ Y_N^k \end{bmatrix}, \quad (13)$$

where the recursions are given by

$$\begin{aligned} X_N^k &= J_N^k,\ Y_N^k = \mathcal{U}_N^k, \\ X_i^k &= J_i^k - \mathcal{B}_U (X_{i+1}^k)^{-1} \mathcal{B}_L, \\ Y_i^k &= \mathcal{U}_i^k - \mathcal{B}_U (X_{i+1}^k)^{-1} Y_{i+1}^k. \end{aligned}$$

Then, (13) is solved recursively from $i = 1$ to N by

$$\Delta\mathcal{V}_{i(nt)}^k = -(X_i^k)^{-1} \left(Y_i^k + \mathcal{B}_L \Delta\mathcal{V}_{i-1(nt)}^k \right), \quad (14)$$

where $\Delta\mathcal{V}_{0(nt)}^k = 0$.

Next, the relationship between $\Delta\mathcal{V}_{(bc)}^k$ and $\Delta\mathcal{V}_{(nt)}^k$ is shown. By combining (7), (9), and (10), the recursion of calculating H_i^k is given by

$$H_i^k = \left(J_i^k - \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_{i+1}^k \end{bmatrix} \right)^{-1} = \left(J_i^k - \mathcal{B}_U H_{i+1}^k \mathcal{B}_L \right)^{-1}. \quad (15)$$

Note that $H_N^k = (J_N^k)^{-1} = (X_N^k)^{-1}$, then from the definition of X_i^k and (15), it follows that $H_i^k = (X_i^k)^{-1}$, $i = 1, \cdots, N$. According to (11),

$$\begin{aligned} &\mathcal{U}_i \left(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k) \right) = \mathcal{U}_i^k - \begin{bmatrix} 0 \\ d_{\lambda_{i+1}}^k \end{bmatrix} \\ &= \mathcal{U}_i^k - \mathcal{B}_U H_{i+1}^k \cdot \mathcal{U}_{i+1} \left(x_i^k, \mathcal{V}_{i+1}^k, \lambda_{i+2}^{k+1}(x_{i+1}^k) \right) \end{aligned}$$

holds. Together with $\mathcal{U}_N \left(x_{N-1}^k, \mathcal{V}_N^k, \lambda_{N+1}^{k+1}(x_N^k) \right) = \mathcal{U}_N^k = Y_N^k$, we can have $\mathcal{U}_i \left(x_{i-1}^k, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k) \right) = Y_i^k$, $i = 1, \cdots, N$. Then, by (12), the recursion of calculating $\Delta\mathcal{V}_{i(bc)}^k$ is given by:

$$\begin{aligned} &\Delta\mathcal{V}_{i(bc)}^k = -H_i^k \cdot \mathcal{U}_i \left(x_{i-1}^{k+1}, \mathcal{V}_i^k, \lambda_{i+1}^{k+1}(x_i^k) \right) \\ &= -(X_i^k)^{-1} (Y_i^k + \mathcal{B}_L \Delta\mathcal{V}_{i-1(bc)}^k),\ i = 1, \cdots, N. \end{aligned} \quad (16)$$

From comparing (14) with (16), and the boundary condition $\Delta\mathcal{V}_{0(bc)}^k = \Delta\mathcal{V}_{0(nt)}^k = 0$, it is clear that $\Delta\mathcal{V}_{i(bc)}^k = \Delta\mathcal{V}_{i(nt)}^k$ for $i = 1, \cdots, N$. Thus, $\mathcal{V}_{(bc)}^{k+1} = \mathcal{V}_{(nt)}^{k+1}$ holds. □

4. PROPOSED PARALLEL METHOD

In the backward correction method, H_i in (7) has to be calculated recursively from $i = N$ to 1 in order to formulate the expression of the corrected λ_{i+1} in (9) for each stage, which is the most computationally expensive part. Although it is not parallelizable, the iteration of each stage is clearly shown in (12). It is possible to have a slower rate of convergence by a relatively coarse estimate of λ_{i+1}^* using a less computationally expensive method. In order to break the recursion, we propose to predict λ_{i+1}^* based on Λ_{i+1}^{k-1} instead of Λ_{i+1}^k, i.e., (9) in the backward correction method can be replaced by

$$\lambda_{i+1}^{k+1}(x_i) = \lambda_{i+1}^k - \Lambda_{i+1}^{k-1}(x_i - x_i^k) - d_{\lambda_{i+1}}^k, \quad (17)$$

for $i = 1, \cdots, N$. The proposed method has the same algorithm as the backward correction method, except that (9) is replaced by (17), i.e., Λ_{i+1}^k is replaced by Λ_{i+1}^{k-1}. To distinguish from the proposed method, let $v_{(bc)}$ be the variable in the backward correction method for a variable v. It can be seen that the proposed method approximates the backward correction method where the approximation is introduced by:

$$h_i^k := \Theta_i^{k-1} - \Theta_{i(bc)}^k,\ i \in \{1, \cdots, N\}, \quad (18)$$

where

$$\Theta_{i(bc)}^k := \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_{i(bc)}^k \end{bmatrix} \in \mathbb{R}^{n \times n}$$

and

$$\Theta_i^{k-1} := \begin{bmatrix} 0 & 0 \\ 0 & \Lambda_i^{k-1} \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

By introducing the approximation, the recursion of calculating H_i^k from $i = N$ to 1 is removed, thus $\{H_i^k\}_{i=1}^N$ can be calculated concurrently by (19) for $i = 1, \cdots, N$.

$$H_i^k = \left(J_i^k - \Theta_{i+1}^{k-1} \right)^{-1} \quad (19)$$

4.1 Parallel algorithm

Calculating $\{H_i^k\}_{i=1}^N$ in parallel leads to an algorithm with complexity $\mathcal{O}(n^3 + Nn^2)$ on N threads. Moreover, it can be further parallelized by arranging the order of computations. After obtaining $\{H_i^k\}_{i=1}^N$, the update can first be done coarsely by $\mathcal{V}_i^{k+1} = \mathcal{V}_i^k - H_i^k \cdot \mathcal{U}_i^k$, which results in Step 1 of Algorithm 2. Notice that the coarse update introduces approximation for both the coupling variables x_{i-1} and λ_{i+1} compared with (12). The correction due to the approximation of λ_{i+1} is conducted in a backward manner, as shown in Step 2 of Algorithm 2. Likewise, the correction due to the approximation of x_{i-1} is conducted in a forward manner, as shown in Step 3 of Algorithm 2. The fully parallelized algorithm is summarized in Algorithm 2, which has a complexity of $\mathcal{O}(n^3 + Nn_x^2)$ on N threads.

Remark 1. Algorithm 2 has to be initialized at every sampling instant. That is, $\mathcal{V}^0$ and $\{\Lambda_i^{k-1}\}_{i=1}^N$ must be provided. The warm-start strategy, which initializes from the previous instant, can be adopted. As for the very first OCP, it can be solved offline.

5. NUMERICAL EXPERIMENT

5.1 NMPC for a quadrotor

In order to demonstrate the computation time and the rate of convergence for the proposed method, reference tracking of a quadrotor is considered. The state vector of the quadrotor is $x = [X, \dot{X}, Y, \dot{Y}, Z, \dot{Z}, \gamma, \beta, \alpha]^T \in \mathbb{R}^9$, where (X, Y, Z) and (γ, β, α) are the position and the angles of the quadrotor, respectively. The input vector is $u = [a, \omega_X, \omega_Y, \omega_Z]^T$, where a represents the thrust and $(\omega_X, \omega_Y, \omega_Z)$ the rotational rates. The following nonlinear model can be found in (Hehn and D'Andrea, 2011):

$$\begin{aligned} \ddot{X} &= a(\cos\gamma \sin\beta \cos\alpha + \sin\gamma \sin\alpha) \\ \ddot{Y} &= a(\cos\gamma \sin\beta \sin\alpha - \sin\gamma \cos\alpha) \\ \ddot{Z} &= a \cos\gamma \cos\beta - g \\ \dot{\gamma} &= (\omega_X \cos\gamma + \omega_Y \sin\gamma)/\cos\beta \\ \dot{\beta} &= -\omega_X \sin\gamma + \omega_Y \cos\gamma \\ \dot{\alpha} &= \omega_X \cos\gamma \tan\beta + \omega_Y \sin\gamma \tan\beta + \omega_Z \end{aligned}$$

The control input is bounded by $[0, -1, -1, -1]^T \leq u \leq [11, 1, 1, 1]^T$. The system starts from the initial

Algorithm 2 Fully parallelized implementation of the k-th iteration for the proposed method

Input: $\mathcal{V}^k$, $\{\Lambda_i^{k-1}\}_{i=1}^N$, $x_0^k = \bar{x}_0$, $\lambda_{N+1}^k = 0$ and $\Lambda_{N+1}^k = 0$.
Output: $\mathcal{V}^{k+1}$ and $\{\Lambda_i^k\}_{i=1}^N$.
Step 1. Coarse update:
for $i = 1$ to N **do in parallel**
Evaluate $\mathcal{U}_i^k$, J_i^k.
Compute $H_i^k = \left(J_i^k - \Theta_{i+1}^{k-1}\right)^{-1}$.
Update $\Lambda_i^k = H_i^k[1:n_x, 1:n_x]$.
Update $\mathcal{V}_i^{k+1} = \mathcal{V}_i^k - H_i^k \cdot \mathcal{U}_i^k$.
end for
Step 2. Backward correction due to the approximation of λ:
for $i = N-1$ to 1 **do**
$d_{\lambda_{i+1}}^{k+1} := \lambda_{i+1}^{k+1} - \lambda_{i+1}^k$.
$\lambda_i^{k+1} = \lambda_i^{k+1} - H_i^k[1:n_x, n-n_x+1:n] \cdot d_{\lambda_{i+1}}^{k+1}$.
end for
for $i = 1$ to $N-1$ **do in parallel**
$\mathcal{V}_i^{k+1}[n_x+1:n,:] = \mathcal{V}_i^{k+1}[n_x+1:n,:]$
$-H_i^k[n_x+1:n, n-n_x+1:n] \cdot d_{\lambda_{i+1}}^{k+1}$.
end for
Step 3. Forward correction due to the approximation of x:
for $i = 2$ to N **do**
$d_{x_{i-1}}^{k+1} := x_{i-1}^{k+1} - x_{i-1}^k$.
$x_i^{k+1} = x_i^{k+1} - H_i^k[n-n_x+1:n, 1:n_x] \cdot d_{x_{i-1}}^{k+1}$.
end for
for $i = 2$ to N **do in parallel**
$\mathcal{V}_i^{k+1}[1:n-n_x,:] = \mathcal{V}_i^{k+1}[1:n-n_x,:]$
$-H_i^k[1:n-n_x, 1:n_x] \cdot d_{x_{i-1}}^{k+1}$.
end for

state $x_0 = [1,0,1,0,1,0,0,0,0,0]^T$ and the state reference is set to $x_{ref} = 0$ from 0 to 10 s and $x_{ref} = [1.5,0,1.5,0,1.5,0,0,0,0,0]^T$ from 10 to 20 s. The input reference is set to $u_{ref} = [g,0,0,0]^T$. The stage cost function is chosen as $L(u,x,p) = \frac{1}{2}(\|x - x_{ref}\|_Q^2 + \|u - u_{ref}\|_R^2)$ with $Q = \mathrm{diag}(10,1,2,1,10,1,1,1,1,1)$ and $R = I_4$. The prediction horizon is $T = 1$ s and is divided into $N = 8, 16, 24, 48, 96$ grids for comparison. The simulation is performed for 20 s and the sampling time is 0.01 s.

5.2 Computation time

The proposed method was implemented using the open-source MATLAB toolkit ParNMPC (Deng, 2018), which can automatically generate parallel C/C++ code with OpenMP (Dagum and Menon, 1998) for NMPC. For comparison, the code generation tool (Houska et al., 2011a) of the ACADO Toolkit (Houska et al., 2011b) and the code generation toolkit AutoGenU (Ohtsuka, 2015) are applied to carry out the closed-loop NMPC simulation. ACADO performs the sequential quadratic programming (SQP) method with Gauss-Newton Hessian approximation (GNHA), where the dense QP solver qpOASES (Ferreau et al., 2014) and the sparse QP solver qpDUNES (Frasch et al., 2015) are employed respectively to solve the underlying QP problems. AutoGenU is based on the C/GMRES method. The simulation was carried out on a desktop computer with dual 2.2 GHz Intel Xeon E5-2650 V4 processors with 24 cores in total with the Hyper-Threading and Turbo Boost features are disabled. All of the comparison scripts can be found on the same GitHub page as ParNMPC.

In principle, the ACADO Code Generation is based on the real-time iteration (RTI) scheme, in which only one SQP iteration is performed. The KKT tolerance is controlled by performing several SQP iterations, and the computation time is the sum of the CPU times returned by the solver. The KKT tolerances of the proposed method and the SQP methods are set to $5 \cdot 10^{-3}$, while in the C/GMRES method, only one iteration is performed per update. It should be noted that these methods do not converge to the same solution even when the same KKT tolerances are achieved because of the differences in handling inequality constraints. Namely, the SQP methods will converge to the optimal solution of the inequality constrained optimization problem, while the proposed method and C/GMRES only approximately take the inequality constraints into account by introducing dummy inputs (Ohtsuka, 2004).

The discretization within the SQP methods is based on the explicit Euler method with multiple shooting. The QP problem in the SQP method solved by qpOASES is condensed and warm-start is enabled. The explicit Euler method is used for the C/GMRES method, and the number of iteration in GMRES is set to 7. Although the SQP methods can be in principle parallelized to a certain degree, e.g., for multiple shooting and the QP solver qpDUNES, they are run in a fully sequential fashion.

The approximation to the backward correction method is defined in (18). Let us denote the relative approximation error by e_h in (20), which measures the distance to Newton's method.

$$e_h := \max_{i \in \{1,\cdots,N\}} \frac{\|h_i\|}{\|\Theta_{i(bc)}\|} \tag{20}$$

The simulation result for the inputs, position, and the relative approximation error e_h after each update by the proposed method with $N = 24$ is shown in Fig. 1. It should be noted that the result obtained by introducing dummy inputs is almost the same as the results of the SQP methods, i.e., active input bound constraints can also be achieved when tracking the position reference. Since the degree of parallelism of the proposed method is N, the program can be run on $\min\{24, N\}$ cores concurrently. Table 1 lists the average computation time per update. Although the computation time of the proposed method is large when running on one core in a fully sequential fashion, a speed-up of 10x at most can be achieved, and the proposed method becomes faster than other methods in the medium and high range of N when running on multiple cores.

As the computation time of the proposed method is in the μs range, the overhead time is non-negligible. Let us denote t_E as the average time per update running on $\min\{24, N\}$ cores in practice, and t_p and t_s as the time of the parallelizable and sequential part running on one core, respectively. The theoretical minimum execution time can be given by $t_A := t_s + t_p/\min\{24, N\}$ according to Amdahl's law. From Table 2, we can see that t_s is less than 1% in the total time elapsed, which indicates that the proposed method is highly parallelizable. In this example, the overhead time accounts for a substantial part of the

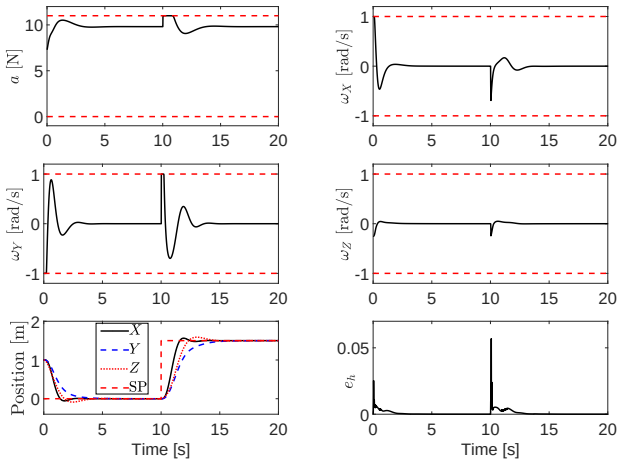

Fig. 1. Time histories of the simulation result by the proposed method (SP is the position reference).

total elapsed time, even more than 60%. One reason is the communication between the dual processors when carrying out the backward and forward corrections steps. This overhead can be reduced by running on a single processor based computer. Also, interruptions by other threads on Windows unbalance the parallel tasks, which also explains that the highest rate of the overhead time is achieved when $N = 24$. The observation of the overhead time shows great potential for the proposed method running on platforms with less overhead time to maximize its efficiency.

Table 1. Average computation time per update $[\mu s]$

N	8	16	24	48	96
Proposed method running on $\min\{24, N\}$ cores	112	139	173	285	454
Proposed method running on one core	455	795	1171	2453	4666
SQP with qpOASES	95	349	866	5920	65745
SQP with qpDUNES	189	361	563	1142	2487
C/GMRES	201	335	470	868	1648

Table 2. Practical and theoretical average computation time per update for the proposed method

N	8	16	24	48	96
t_p $[\mu s]$	452	790	1163	2433	4623
t_s $[\mu s]$	2.8	5.4	8.1	20.2	43.3
$t_s/(t_s + t_p) \cdot 100$ [%]	0.61	0.67	0.69	0.82	0.93
t_E $[\mu s]$	112	139	173	285	454
t_A $[\mu s]$	59	55	57	122	236
$(t_E - t_A)/t_E \cdot 100$ [%]	47	61	67	57	48

5.3 Number of iterations under different tolerances

The proposed method was compared with the backward correction method (Newton's method) and the SQP method with GNHA (qpOASES was used) to investigate the number of iterations under different tolerances. The numbers of iterations shown in Fig. 2 are filtered by a 10-th order moving average filter to remove the jitter that arose on the critical point of satisfying the KKT tolerance, and thus, the numbers of iterations become distinguishable for different methods. It is clear that the proposed method can converge to the specified tolerance in only several iterations and is faster than the SQP method with GNHA. The proposed method is slightly slower than Newton's method, which can be explained from the small relative approximation error e_h in Fig. 1.

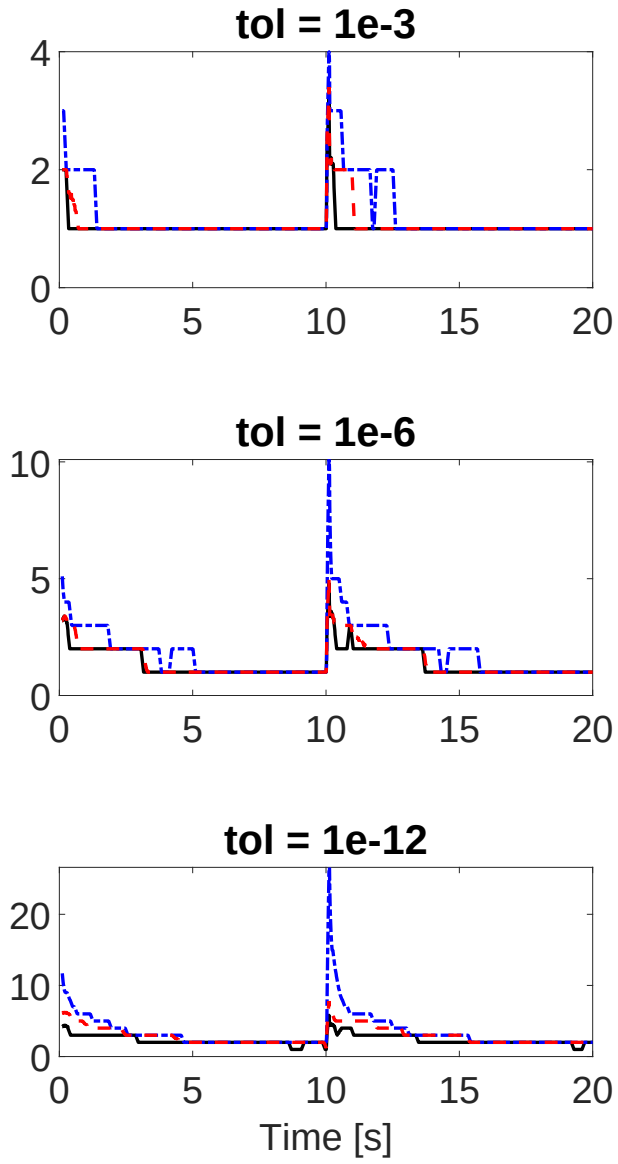

Fig. 2. Filtered number of iterations under different tolerances (black solid line: Newton's method; blue dash-dot line: SQP with GHNA; red dashed line: proposed method).

The rate of convergence is quadratic for Newton's method, and linear (Bock, 1983) for the SQP method with GNHA in general. The proposed method exhibits a faster linear rate of convergence in this example.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a novel Newton-type parallel algorithm for NMPC. The computational complexity of

the proposed method is $\mathcal{O}(n^3+Nn_x^2)$ on N threads. Since it uses multiple-shooting and implicit integrator inherently, it is expected to be capable of dealing with systems that are stiff, unstable, or highly nonlinear. The numerical results show that the proposed method converges fast. However, the rate of convergence is not characterized and remains as one of the main subjects for future work. Another subject is the integration of globalization techniques, such as the line-search method and the trust-region method, to increase the numerical robustness of the proposed method.

REFERENCES

Amdahl, G.M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the Sprint Joint Computer Conference*, 483–485. Atlantic City, USA.

Bock, H.G. (1983). Recent advances in parameter identification techniques for ODE. In P. Deuflhard and E. Hairer (eds.), *Numerical Treatment of Inverse Problems in Differential and Integral Equations*, volume 2 of *Progress in Scientific Computing*, 95–121. Boston: Birkhäuser.

Dagum, L. and Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1), 46–55.

Deng, H. (2018). ParNMPC Homepage. URL `https://github.com/deng-haoyang/ParNMPC`.

Diehl, M., Bock, H.G., Schlöder, J.P., Findeisen, R., Nagy, Z., and Allgöwer, F. (2002). Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4), 577–585.

Ferreau, H.J., Kirches, C., Potschka, A., Bock, H.G., and Diehl, M. (2014). qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4), 327–363.

Frasch, J.V., Sager, S., and Diehl, M. (2015). A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computation*, 7(3), 289–329.

Graichen, K. and Käpernick, B. (2012). A real-time gradient method for nonlinear model predictive control. In T. Zheng (ed.), *Frontiers of Model Predictive Control*. InTech.

Hehn, M. and D'Andrea, R. (2011). A flying inverted pendulum. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 763–770. Shanghai, China.

Houska, B., Ferreau, H., and Diehl, M. (2011a). An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica*, 47(10), 2279–2285.

Houska, B., Ferreau, H.J., and Diehl, M. (2011b). ACADO toolkit–An open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3), 298–312.

Kalmari, J., Backman, J., and Visala, A. (2015). A toolkit for nonlinear model predictive control using gradient projection and code generation. *Control Engineering Practice*, 39, 56–66.

Kouzoupis, D., Ferreau, H.J., Peyrl, H., and Diehl, M. (2015). First-order methods in embedded nonlinear model predictive control. In *Proceedings of the European Control Conference*, 2617–2622. Linz, Austria.

Kouzoupis, D., Quirynen, R., Houska, B., and Diehl, M. (2016). A block based ALADIN scheme for highly parallelizable direct optimal control. In *Proceedings of the 2016 American Control Conference*, 1124–1129. Boston, USA.

Nielsen, I. and Axehill, D. (2015). A parallel structure exploiting factorization algorithm with applications to model predictive control. In *Proceedings of the 54th IEEE Conference on Decision and Control*, 3932–3938. Osaka, Japan.

Ohtsuka, T. (2004). A continuation/GMRES method for fast computation of nonlinear receding horizon control. *Automatica*, 40(4), 563–574.

Ohtsuka, T. (2015). A tutorial on C/GMRES and automatic code generation for nonlinear model predictive control. In *Proceedings of the European Control Conference*, 73–86. Linz, Austria.

Xu, F., Chen, H., Gong, X., and Mei, Q. (2016). Fast nonlinear model predictive control on FPGA using particle swarm optimization. *IEEE Transactions on Industrial Electronics*, 63(1), 310–321.

Yang, X. and Biegler, L.T. (2013). Advanced-multi-step nonlinear model predictive control. *Journal of process control*, 23(8), 1116–1128.

Zavala, V.M. (2016). New architectures for hierarchical predictive control. *IFAC-PapersOnLine*, 49(7), 43–48.